GENERIC NETWORK PROTOCOL LAYER WITH SUPPORTING DATA STRUCTURE

TECHNICAL FIELD

The present invention is generally related to computer network communications protocols. More particularly, the present invention relates to a method, program product and data structure for the implementation of a generic network protocol class usable by multiple network protocols running on a single computer system.

BACKGROUND OF THE INVENTION

Networks provide computers with the basic ability to transfer data from one computer to another computer. However, in order to be able to communicate with each other, the computers attached to the network require sets of rules for communication called "protocols." These protocols specify how the computers on the network interact and exchange messages. In most cases, the protocols provide rules for, among other things, the formatting of messages and the handling of errors.

In order to simplify the design and implementation of protocols and provide a computer with the ability to communicate over one or more networks to one or more different computers, programmers have typically used a set of separate, independent protocols where each protocol has different responsibilities. This set of protocols is called a "protocol suite" and covers all forms of a computer's network communications as needed.

A layering model is the most common way to divide a protocol suite into sub-parts and describe each part individually. The most well know layering model for network communications is the seven layer Open Systems Interconnect model introduced by the International Organization for Standardization more than twenty years ago (the "OSI Model"). The basis of this model is illustrated in Table 1.

TABLE 1

Layer	Responsibilities
1. Physical	Specifies the electrical and mechanical connections to the network
2. Data Link	The data link layer splits data into frames for sending on the physical layer and receives acknowledgment frames. It performs error checking and re-transmits frames not received correctly. It provides an error-free virtual channel to the network layer. The data link layer is split into an upper sub-layer, Logical Link Control (LLC), and a lower sub-layer, Media Access Control (MAC).
3. Network	The network layer determines routing of packets of data from sender to receiver via the data link layer and is used by the transport layer. The most common network layer protocol is Internet Protocol (IP).
4. Transport	The transport layer determines how to use the network layer to provide a virtual error-free, point to point connection so that host A can send messages to host B and they will arrive un-corrupted and in the correct order. It establishes and dissolves connections between hosts. An example transport layer protocol is Transmission Control Protocol (TCP).
5. Session	The session layer uses the transport layer to establish a connection between processes on different hosts. It handles security and creation of the session.

6. Presentation	Performs functions such as text compression, code or format conversion to try to smooth out differences between hosts. Allows incompatible processes in the application layer to communicate via the session layer.
7. Application	This layer handles issues like network transparency, resource allocation and problem partitioning. The application layer is concerned with the user's view of the network (e.g. formatting electronic mail messages). The presentation layer provides the application layer with a familiar local representation of data independent of the format used on the network.

In the foregoing table, the application layer is typically referred to as the "highest" protocol layer and the physical layer is typically referred to as the "lowest" protocol layer.

Under the OSI Model, the sending of data usually is initiated by the application layer which (a) specifies the original data to be sent, (b) creates a data packet by adding to the original data any header or checksum data required by the application layer and then (c) passes the resulting data packet to the presentation layer. The presentation layer then (i) processes the data packet and creates one or more new data packets from the data packet passed to it by the application layer, (ii) adds to each of the new data packets any header or checksum data required by the presentation layer and then (iii) passes the new data packets to the session layer. Processing the data packet may include dividing the packet into smaller pieces in accordance with the processing or memory limits of the current layer. This cycle of (i) processing a passed data packet and creating one or more new data packets from the passed data packet, (ii) adding to each of the new data packets any header or checksum data required by the acting protocol layer and then (iii) passing the new data packets to the next lower protocol layer is then repeated by the session layer and each protocol layer below the session layer until the data packet reaches the physical layer. Once a data packet reaches the physical layer, it is placed onto the physical network for delivery to the receiving computer.

Upon receipt of a data packet from the physical network by the receiving computer's physical layer, the data packet is passed to the data link layer which then (a) processes the data packet and strip off any headers and checksum data added to the data packet by the sending computer's data link layer, (b) combines (as applicable) into a new data packet the processed and stripped data packet and zero or more previously processed and stripped data packets and then (c) passes the resulting data packet to the network layer. This cycle of (i) processing the passed data packet and stripping off any headers and checksum data added to the data packet by the same protocol layer of the sending computer and (ii) combining (as applicable) into a new data packet the processed and stripped data packet with zero or more previously processed and stripped data packets and then (ii) passing the resulting data packet to the next higher protocol layer continues until the data packet reaches the application layer. The application layer then strips off any headers and checksum data added to the original data by the sending computer's application layer and processes the data as necessary.

As shown by the foregoing, (i) during the sending of data, each protocol layer copies the data packet passed to it by a higher protocol level along with any header and checksum data added by the previous protocol layer and (ii) during the receipt of data, each protocol layer copies the received data less any headers and checksums added to the data by such protocol layer on the sending side. In addition, on most computer systems used today, each protocol layer is developed independently by a separate group of programmers with no sharing of code between protocol layers. Therefore, because of this copying of data and no sharing of code between protocol layers, there are significant inefficiencies in processing and memory usage during the sending and receipt of data by computers over a network. The reduction of these inefficiencies and use of memory becomes more important

as multiple protocol layers are included in "embedded" applications (e.g. home appliances, monitoring equipment, etc.) which will, because of various cost pressures, be implemented using less costly microprocessors having significantly less processing power and memory than stand alone computer systems.

SUMMARY OF THE INVENTION

It is a principal object of the invention to provide a method that allows the sharing of code between separate protocol layers. It is another object of this invention to provide a method and structure that eliminates the need for the copying of data between protocol layers.

According to the invention, the foregoing objects are accomplished by the creation of a generic protocol layer class (GPLC) having the two key methods (in addition to such other methods as are necessary) "send" and "receive" and a common data buffer in which the sent or received data is placed and acted upon by each protocol layer implemented with the GPLC. Instead of copying the data packet passed to it by a higher or lower protocol layer implemented with the GPLC, a protocol layer implemented with the GPLC acts upon the common data buffer by moving a "start" pointer and an "end" pointer along the data contained in the common data buffer prior to invoking the next higher or lower protocol layer implemented with the GPLC.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is better understood by reading the following detailed description of the preferred embodiment in conjunction with the accompanying drawings, wherein:

Fig. 1 depicts the logical flow of the "send" method of the GPLC.

Fig 2 depicts the logical flow of the "receive" method of the GPLC.

Fig. 3 depicts the contents of the common data buffer during the sending of data following the placement of the data being sent into the common data buffer by the first GPLC based protocol layer.

Fig. 4 depicts the contents of the common data buffer during the sending of data following the extraction of the first slice and the addition of header and checksum information by the first GPLC based protocol layer.

Fig. 5 depicts the contents of the common data buffer during the sending of data following the addition of header and checksum information by the second GPLC based protocol layer.

Fig. 6 depicts the contents of the common data buffer during the sending of data following the addition of header and checksum information to the second slice by the first GPLC based protocol layer.

Fig. 7 depicts the contents of the common data buffer during the sending of data following the addition of header and checksum information by the second GPLC based protocol layer.

Fig. 8 depicts contents of the common data buffer upon the initiation of the receipt of data by the first GPLC based protocol layer.

Fig. 9 depicts contents of the common data buffer during the receipt of data following the placement of the data into the common data buffer by the second GPLC based protocol layer.

Fig. 10 depicts the contents of the common data buffer during the receipt of data following the stripping by the second GPLC based protocol layer of header and checksum information placed on such data by the equivalent protocol layer during sending.

Fig. 11 depicts the contents of the common data buffer during the receipt of data following the stripping by the first GPLC based protocol layer of header and checksum information placed on such data by the equivalent protocol layer during sending.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the preferred embodiment of the invention, a basic implementation of the GPLC for two protocol layers is considered. However, the use of only two protocol layers in the description of the preferred embodiment of the GPLC as shown in Figs. 3 through 11 should in no way be considered a limitation on the number of protocol layers able to be implemented with the GPLC. In addition, for the purposes of understanding the preferred embodiment of the GPLC, it should be noted that the GPLC is not limited only to the implementation of one or more different protocol layers, but may also be used to implement multiple sub-layers within each protocol layer (i.e. multiple applications within the application layer that have a communications hierarchy). In fact, in some instances, it may be practical only to implement such sub-layers with the GPLC as the other protocol layers may be readily and economically available from third-party sources. Therefore, for the purposes of the

description of the preferred embodiment, each reference to "protocol layer" should be read as "protocol layer or sub-layer" unless expressly specified otherwise.

The present invention is applicable to computer systems that communicate with other computer systems over a computer communications network. As such, there are no restrictions on the type of network involved or on the specific communications protocols employed. For example, the invention is equally applicable to both conventional "wired" computer networks and "wireless" computer networks. It can be utilized in networks including Transport Control Protocol/ Internet Protocol (TCP/IP) networks, the Asynchronous Transport Mode (ATM) networks, Frame Relay networks, Integrated Services Digital Networks (ISDN) networks, the Global System for Mobile Communication (GSM) network, Ethernet local area networks (LANs), token ring LANs, and any other network having a layered network architecture. The network connection can include an Ethernet or token ring LAN adapter, a Personal Computer Memory Card International Association (PCMCIA) card or a wireless (cellular) modem card.

For the preferred embodiment, an object class is used that includes:

(1) a "send" method;

(2) a "receive" method;

(3) a common data buffer used by all members of the object class;

(4) and such other methods and data structures as are common to and required by each of the network protocol layers to be implemented with the GPLC.

Under this object class, the "send" and "receive" methods act upon the common data buffer defined for the class as explained in more detail below. In order to communicate with another protocol level

implemented with the object class, the "send" and "receive" methods respectively invoke the "send" and "receive" methods of the next lower protocol layer implemented with the object class, passing only pointers to the "start" and "end" of the data stored in the common data buffer.

As described herein, the "send" and "receive" methods of the preferred embodiment of the GPLC differ significantly from the prior art. Under the prior art, each protocol layer is programmed independently from the other protocol layers with which it communicates. When communicating with another protocol layer, the "send" and "receive" methods and procedures of the sending or receiving protocol layer pass to the next protocol layer a copy of the data being sent or received. The next protocol layer then takes this copy of the data, processes it, adds its own headers and checksum information to the data and passes a copy of the entire bundle to the next protocol layer as applicable. Under the prior art, the whole data gets copied multiple times as it is handled by the various protocol layers. This multiple copying of the data results in an inefficient use of memory and processing resources. With the GPLC, the copying of data only occurs at the interface boundaries between protocol layers implemented with the GPLC and those that are not, thereby reducing the amount of memory and processing time required for the sending and receipt of data. This reduction in resource use is especially useful in embedded applications (i.e. home appliances, monitoring equipment, etc.) where, in order to minimize costs, memory and processing resources are intentionally limited.

Referring now in more detail to the drawings in which like numerals refer to like parts throughout several views, Fig. 1 shows the general logic flow for the sending of data using the "send" method of the GPLC. The logic flow demonstrated by Fig 1 begins with the invoking of "send" method 10 by a higher level of protocol layer's send method or procedure and passing to

"send' method 10 of any required data or other information. The first step 12 adjusts the "start" and "end" pointers of the common data buffer accordingly in order to extract an appropriately sized slice for sending. (For the purposes of Fig. 1 and other figures and descriptions related to the preferred embodiment, the term "slice" is used to refer in a generic manner to a portion of data, which may be, for example and without limitation, a packet, a block or a physical record depending on the application.) This slice, depending on the size of the data being sent as compared to the maximum block size of data able to handled by the protocol layer implemented by "send" method 10, may be all or only a portion of the data passed to "send" method 10 by the next higher protocol layer. Step 14 then adds to the data the headers and checksums of the protocol layer implemented by "send" method 10. Step 16 invokes the next lower protocol layer's "send" method 20, passing to such method any required data or other information.

The first step 22 of "send" method 20 adjusts the "start" and "end" pointers of the common data buffer accordingly in order to extract an appropriately sized slice for sending. This slice, depending on the size of the data, headers and checksums passed by "send" method 10 as compared to the maximum block size of data able to handled by the protocol layer implemented by "send" method 20, may be all or only a portion of the data, headers and checksums passed by "send" method 10. Step 24 then adds to the data the headers and checksums of the protocol layer implemented by "send" method 20. Step 26 invokes the next lower protocol layer's "send" method or procedure, passing to such method or procedure any required data or other information. Upon the exiting of the next lower protocol layer's "send" method or procedure and return of control to "send" method 20, step 28 then exits, thereby returning control to "send" method 10 and passing to "send" method 10

any required data or other information. Step 18 then exits, thereby returning control to the higher level protocol layer that invoked it and passing to such higher level protocol layer any required data or other information.

Fig. 2 shows the general logic flow for the receipt of data using the "receive" method of the GPLC. The logic flow begins with the invoking of "receive" method 50 by a higher level method or procedure that is ready to receive data (i.e. this higher level thread or process is ready to wait for the receipt of data by, among other things, suspending execution until data arrives). The first step 52 of "receive" method 50 invokes the next lower protocol layer's "receive" method 60, passing to such method any required information (e.g. the location of the receiving buffer for the data). The first step 62 of "receive" method 60 then invokes the next lower protocol layer's "receive" method or procedure, passing to such method or procedure any required information.

Upon return from the "receive" method or procedure of the next lower level protocol layer, step 64 processes the headers and checksums added by the "send" method of the same level protocol layer that originally sent the data. Step 66 strips of any such headers and checksums from the data. Step 68 exits, thereby returning control to the higher level protocol layer method 50 and passing to such method any required data or other information.

Upon return from "receive" method 60, step 54 processes the headers and checksums added by the "send" method of the same level protocol layer that originally sent the data. Step 56 strips of any such headers and checksums from the data. Step 58 then exits, thereby returning control to the higher level protocol layer method or procedure that invoked it and passing to such method any required data or other information.

It should be noted that, based upon the particular environment and operation of a particular implementation of the GPLC, the data being sent may, at one or more protocol layer levels, be broken into smaller packets during the sending process, with reassembly of these packets taking place during the receiving process. This breaking up and reassembly may be necessitated by, among other things, the differences in block sizes between specific implementations of the various protocol layers, hardware limitations, and/or network limitations. In addition, there is no requirement that the data be reassembled on the receiving side at the same protocol layer as it was broken up on the sending side.

Figs. 3 through 7 depict the state of the common data buffer during various stages of the sending of data where only two protocol layers are implemented using the GPLC. Figs. 8 through 11 depict the state of the common data buffer during various stages of the receipt of data where only two protocol layers are implemented using the GPLC. For the purposes of Figs. 3 through 11, what is referred to as the "first" protocol layer is the highest level protocol layer implemented with the GPLC, with the "second" protocol layer being the lowest level protocol layer implemented with the GPLC.

Fig. 3 depicts the state of the common data buffer 100 during the sending of data following the placement of the original data 120 to be transferred into the common data buffer 100 by the "send" method of the first protocol layer. Typically, the common data buffer 100 has unused memory space 110 and 130 on each side of the original data 120, but this is not required. The start pointer 140 is set to the location of the first byte of the original data 120 and the end pointer 150 is set to the location of the last byte of the original data 120.

Fig. 4 depicts the state of the common data buffer 100 during the sending of data following the extraction of the first slice by the "send" method of the first protocol layer and the addition by such "send" method of the protocol layer's headers and checksums 160. The start pointer 140 is moved to the location of the first byte of the header 160. The end pointer 150 is moved to the location of the byte in the original data 120 that is the end of the first slice.

Fig. 5 depicts the state of the common data buffer 100 during the sending of data following the addition by the "send" method of the second protocol layer of the protocol layer's headers and checksums 170. The start pointer 180 is moved to the location of the first byte of header 170. The end pointer 190 remains at the location of the byte in the original data 120 that is the end of the first slice of data. It should be noted that each "send" method maintains its own local start pointer and end pointer that are initialized to the values of the invoking protocol layer's start pointer and end pointer when the "send" method is invoked by the higher level protocol layer. The reason for this is that the active start pointer and end pointer automatically are set to the proper position in the common data buffer upon the exit and return of a "send" method invoked by the "send" method of a higher level protocol layer therefore eliminating the need for each "send" method to recalculate the position of each pointer upon the exit and return of the invoked lower level protocol's "send" method.

Fig. 6 depicts the state of the common data buffer 100 during the sending of data following the extraction of the second slice by the "send" method of the first protocol layer and the addition by such "send" method of the protocol layer's headers and checksums 200. The start pointer 140 is

moved to the location of the first byte of the header 200. The end pointer 150 is moved to the location of the byte in the original data 120 that is the end of the second slice.

Fig. 7 depicts the state of the common data buffer 100 during the sending of data following the addition by the "send" method of the second protocol layer of such protocol layer's headers and checksums 210. The start pointer 180 is moved to the location of the first byte of the header 210. The end pointer 190 is moved to the location of the byte in the original data 120 that is the end of the first slice.

Fig. 8 depicts the state of the common data buffer 500 during the receipt of data upon the initiation of the receipt processes by the "receive" method of the first protocol layer. The start pointer 520 and end pointer 530 are set to the same byte of unused memory space 510 located within the common data buffer 500.

Fig. 9 depicts the state of the common data buffer 500 during the receipt of data following the receipt by the second protocol layer of the received data 550 and attached headers 560 and 570 and placement of such data and headers into the common data buffer 500 by the "send" method of the second protocol layer operating on the remote machine. The start pointer 580 is set to the location of the first byte of header 570 (the last header to be added onto the received data during the send process). The end pointer 590 is set to the location of the last byte of the received data 550.

Fig. 10 depicts the state of the common data buffer 500 during the receipt of data following the stripping of header 570 by the "receive" method of the second protocol layer. The start pointer 580 is moved to the location of the first byte of header 560 (the first header to be added onto the

received data during the send process). The end pointer 590 remains set to the location of the last byte of the received data 550.

Fig. 11 depicts the state of the common data buffer 500 during the receipt of data following the stripping of header 560 by the "receive" method of the first protocol layer. The start pointer 520 is moved to the location of the first byte of the received data 550. The end pointer 530 remains set to the location of the last byte of the received data 550. In the preferred implementation, when practical, any needed data reassembly (i.e. from multiple packets) is performed only at the highest application layer as this reduces the amount of copying and moving of data that is required.

The present invention can be realized in software or a combination of hardware and software. Any kind of computer system or other apparatus adapted for carrying out the methods described herein is suited. A typical combination of hardware and software could be a general purpose computer system with a computer program that, when loaded and executed, controls the computer system such that it carries out the methods described herein. The present invention can also be embedded in a computer program product, which comprises all the features enabling the implementation of the methods described herein, and which, when loaded in a computer system, is able to carry out these methods.

Computer program instructions or computer program in the present context mean any expression, in any language, code or notation, of a set of instructions intended to cause a system having an information processing capability to perform a particular function either directly or after either or both of the following occur: a) conversion to another language, code or notation; b) reproduction in a different material form.

Those skilled in the art will appreciate that many modifications to the preferred embodiment of the present invention are possible without departing from the spirit and scope of the present invention. In addition, it is possible to use some of the features of the present invention without the corresponding use of the other features. Accordingly, the foregoing description of the preferred embodiment is provided for the purpose of illustrating the principles of the present invention and not in limitation thereof, since the scope of the present invention is defined solely by the appended claims.